Research on free graphics-related software

On this page, information is collected that could help solving graphics issues in Replicant (see #1539). Besides evaluating free implementations that are relevant for currently supported devices, other implementations should also be listed if they are useful for potential future target devices.

External resources:
Free and open-source graphics device drivers - Wikipedia
Mobile drivers - Debian Wiki

Multiple backends

Replicant 6.0 0004 RC1 includes a mechanism that allows choosing between different OpenGL ES implementations for each app or process. This came into place as means to achieve a balance between fully-compliant but slower implementations (e.g. llvmpipe) and non-compliant but fast implementations (e.g. libagl).

A similar mechanism may be used in future Replicant versions, to take advantage of GPU backed implementations, even if they do not achieve full OpenGL ES compliance.

Software rendering

Software rendering uses the CPU and not a dedicated graphics processor for graphics rendering. It is slower than per-GPU implementations and is mostly used as a fallback, when GPU acceleration is not available. An advantage is that the same software renderer can work across many different types of hardware. Therefore, improving a software renderer benefits many different devices, regardless of the SoC and graphics unit that they have. Furthermore, a software renderer doesn't require a kernel driver. This makes it easier to work on mainline Linux kernel support for a device, until the graphics driver is in mainline.

OpenGL ES software renderers

libagl

- Source code: https://git.replicant.us/replicant/frameworks_native/tree/opengl/libagl

libagl is the fastest software renderer available for Replicant devices. It is used by default on all Replicant-supported devices up until Replicant 6.0 0003 (0004 switched to llvmpipe).

libagl was developed specifically for Android and it is part of the AOSP source code. The renderer includes optimizations for ARM via libpixelflinger and codeflinger that do JIT compilation into platform optimized code. Development ceased in 2013 and no work was done to support newer OpenGL ES (GLES) versions (which causes #705).

Until ca. 2011 (Android 4.0), another library with the name libagl2 existed and surfaceflinger2 was developed based on Mesa. The source code was later removed and no further development is known. At the time, the work was done to support a newer GLES version for the software renderer. It was abandoned later, probably when Google made it mandatory for Android 4.0 and later devices to pack their own hardware GPU with OpenGL ES 2.0 support. It is questionable if it is worth it to port the old libagl2 library to a recent Replicant version, also given that we would be the only ones using and maintaining the code.

Mesa's llvmpipe

- Replicant llvmpipe support source code: https://git.replicant.us/replicant/external_mesa3d
- Replicant documentation: Graphics
- Upstream documentation: https://www.mesa3d.org/llvmpipe.html
- Upstream performance improvement documentation: https://www.mesa3d.org/perf.html

Mesa's llvmpipe is the default software renderer for Replicant since version 6.0 0004. It has a GLES implementation that is more complete than libagl, although slower when used by some system components (e.g. SurfaceFlinger).

Besides llvmpipe, Mesa has two other software rasterizers: swrast/swr and softpipe, but both are of no interest. swrast's GLES implementation is incomplete and this driver is mostly deprecated in favor of those built upon Gallium (softpipe and llvmpipe). softpipe on the other hand, is as complete as llvmpipe but is slower than it.

The Android-x86 project is using llvmpipe. A few of their Android-specific framework patches are applied in Replicant 6.0. Their Mesa source code fork is also used in Replicant 6.0. A lot of porting work of llvmpipe to Android was done by Jide while Intel is contributing

as well. So there is an interest from different parties to have llvmpipe working on Android. Android patches are upstreamed to mainline Mesa.

llvmpipe is still not ported to ARM which makes it slow. Also for Android, it is mostly used on the x86 platform in other projects. See #705 for more information. Optimizing llvmpipe for ARM seems currently the most promising approach to fix graphics-related issues with Replicant.

Tuning llvmpipe

The following environment variables might speed up llvmpipe somewhat:

```
LP_PERF=no_mipmap,no_linear,no_mip_linear,no_tex,no_blend,no_depth,no_alphatest MESA_NO_DITHER=1
```

LP_PERF seems to be undocumented; MESA_NO_DITHER is documented in the Mesa performance tips. According to adjtm, MESA_NO_DITHER=1 improved glxgears performance by 3% on a GNU/Linux PC without breaking any apps that they tested. LP_PERF=no_mipmap,no_linear,no_mip_linear also didn't break any apps on GNU/Linux in tests but there was no noticeable increase in performance. no_tex,no_blend,no_depth,no_alphatest broke rendering of all tested apps in GNU/Linux.

To test with those environment variables on a per-app basis, you can disable SELinux, and then run the following, substituting the name of the app (here info.guardianproject.orfox) for the one you wish to test; then run the app from the launcher:

```
setprop wrap.info.guardianproject.orfox "LP_PERF=no_mipmap,no_linear,no_mip_linear,no_tex,no_blend
,no_depth,no_alphatest MESA_NO_DITHER=1"
```

Unfortunately, wrap seems to crash Replicant sometimes. There was allegedly a fix at https://android-review.googlesource.com/c/platform/frameworks/base/+/318859 .

Setting those environment variables globally might be possible by editing https://git.replicant.us/replicant/system_core/tree/rootdir/init.environ.rc.in .

Benchmarking performance in real-world apps might be feasible via this script: https://github.com/romannurik/env/blob/master/bin/android-fps-count

SwiftShader

- Source code: https://swiftshader.googlesource.com/SwiftShader

Google released SwiftShader as free software in mid 2016. It supports x86 and ARM architectures with SDIV/UDIV support. It is used in the Chromium project but also with Android, for example in the Android-x86 project. Swiftshader doesn't seem to depend on any external libraries besides what are provided in its Git repository, therefore it is very easy to compile it and use it as a software renderer for Replicant. All the Android build files are provided so you just need to the add the following packages to PRODUCT_PACKAGES in order to be able to use it:

- libGLESv2_swiftshader
- libEGL_swiftshader

SwiftShader features a broadly compliant GLES implementation like llvmpipe.

Vulkan software renderers

SwiftShader

Apart from GLES, SwiftShader also supports Vulkan. Actually, Vulkan is SwiftShader's main focus now.

Lavapipe

- Source code: https://gitlab.freedesktop.org/mesa/mesa/-/tree/master/src/gallium/frontends/lavapipe

Lavapipe, previously known as Vallium, is a Vulkan software renderer based on Mesa's llvmpipe and Gallium. It is merged into mainline Mesa and being continuously improved by Dave Airlie, which is leading the effort single-handedly.

It is worth to follow Lavapipe's progress and later test it on Replicant 11.

Kazan

- Source code: https://salsa.debian.org/Kazan-team/kazan

Kazan is another work-in-progress Vulkan software renderer. It is an independent stack, not relying on any underlying graphics library. However, much like Vallium and SwiftShader, it uses LLVM for its shader compiler. Kazan is programmed in Rust.

kms_swrast

kms_swrast is a Mesa driver, built upon Gallium, that uses DRM nodes for memory allocation and to present images on the display, but does the actual OpenGL rendering through a Mesa software renderer such as llvmpipe or softpipe.
It may allow noticeable performance improvements by avoiding expensive memory copy operations between the software renderer and DRM.

Exynos based devices have a working free-software DRM driver that can be used with kms_swrast. Devices that do not have DRM driver can still benefit from kms_swrast by usage of the VGEM (Virtual GEM) driver.

Pixman

- Source code: https://cgit.freedesktop.org/pixman/

  Pixman is a low-level software library for pixel manipulation, providing features such as image compositing and trapezoid rasterization.

It is highly optimized for ARM processors and has a fast path for ARM NEON. It may be worthwhile to write an OpenGL ES backend for Replicant that detects 2D operations and translates them to Pixman. This could provide a considerable performance improvement over llvmpipe or SwiftShader and, if complete enough, could even replace them.

On the other hand, writing such translation layer may prove to be an enormous task, as the OpenGL ES 2.0 API is quite extensive. In this scenario we can still benefit from Pixman by reproducing it's ARM NEON fast paths on llvmpipe.

Per-GPU implementations

In the following, free software implementations are listed that should make it possible to use the respective GPU with free software.

ARM Mali-4xx (Utgard) with Lima

Supported devices that use Mali 400: Galaxy S 2, Galaxy S 3, Galaxy S 3 4G, Galaxy Note, Galaxy Note 2, Galaxy Note 8.0

- Project page: https://gitlab.freedesktop.org/lima/web/-/tree/master
- Source code: https://gitlab.freedesktop.org/mesa/mesa/-/tree/master/src/gallium/drivers/lima

Lima is now merged into Mesa and should be compatible with GLES 2.0. Supported extensions are listed at docs/features.txt and can be visualized with https://mesamatrix.net. GLES 3 is not planned because Utgard's "programmable pipeline" is not much more flexible than a fixed function pipeline, and cannot cope with the GLES 3 features which usually require a unified shader model.

Lima saw first light with the reverse-engineering efforts by Luc Verhaegen, that produced an experimental driver with it's original page at
https://limadriver.org . Luc's development stopped around 2014, but in 2017 Qiang Yu took on the task and started development on top of Mesa's Gallium3D driver as reported by Phoronix.
The code was then hosted at freedesktop.org's GitLab and later merged into mainline Mesa. It gets contributions by many developers besides Qiang Yu.

Apart from the driver in Mesa, Lima also has a corresponding driver in the Linux kernel.

Imagination PowerVR

Supported devices that use PowerVR SGX540: Nexus S, Galaxy S, Galaxy Nexus, Galaxy Tab 2 7.0, Galaxy Tab 2 10.1
Supported devices that use PowerVR SGX530: GTA04

No free software driver is available.

There is currently an effort by the OpenPVRSGX Linux Driver Group to build a device driver for the PVR/SGX5 architecture that is compatible with mainline Linux. They are building it out of old drivers released under GPL which are now incompatible with mainline. Do note that this project is just about the kernel driver, which just initializes the hardware and feeds it the command stream. It still relies on the proprietary user-space driver to implement OpenGL, compile shaders, and generate the command stream.

A reverse-engineering project existed, which aimed to replace the user-space driver with a free-software alternative built upon Mesa:

- Website: http://powervr.gnu.org.ve/doku.php
- Mailing list: http://listas.gnu.org.ve/listinfo/powervr-devel
- Libreplanet group: https://libreplanet.org/wiki/Group:PowerVR_drivers

Despite initial reverse-engineering progress until 2013, no further development took place and the project's website is now offline. Fortunately it can still be accessed through the Internet Archive's Wayback Machine: https://web.archive.org/web/20170923050320/http://powervr.gnu.org.ve/doku.php

In November 2014 the proprietary user-space driver source code for PowerVR SGX Series 5 and Series 5XT was leaked which created even more roadblocks for an already difficult reverse engineering effort.
As of February 2019 there was an attempt by Philipp Rossak to kickstart a clean room reverse engineering effort, but no further news were heard.

ARM Mali-T6xx/T7xx/T8xx (Midgard) and G7x (Bifrost) with Panfrost

Not used by a supported device.
Used on several Samsung Galaxy S series phones that are now supported by LineageOs and may be potential targets for Replicant ports: S6 (Mali T760MP8), S7 (Mali-T880 MP12) and S9 (Mali-G72 MP18).

- Project page: https://panfrost.freedesktop.org/
- Source code: https://gitlab.freedesktop.org/mesa/mesa/-/tree/master/src/gallium/drivers/panfrost

Panfrost is another reverse-engineered driver for ARM Mali GPUs based on Mesa's Gallium3D, but aimed at the latest architectures (Midgard and Bifrost). It is merged into Mesa and under active development with a strong community (as of February 2021).
Supported extensions are listed at docs/features.txt and can be visualized with https://mesamatrix.net. It currently supports OpenGL ES 3.0 and desktop OpenGL 3.1 on both architectures (Midgard and Bifrost). According to the developers, Vulkan support should be achievable on both architectures.
Panfrost's main focus are the ARM SoCs present on some laptops and single-board computers (SBC), but it is intended to work on as many SoCs as possible to make everyone's lives easier.
Panfrost is developed by Alyssa Rosenzweig and Lyude Paul, and recently attracted other contributors such as Tomeu Vizoso and Rob Herring.
Development can be followed on Alyssas's blog.

Qualcomm Adreno with freedreno

Not used by any supported device.

- Wiki: https://github.com/freedreno/freedreno/wiki

freedreno is actively developed. Linux kernel component is available in mainline since version 3.12. It needs to be investigated how well freedreno could work on potential target devices. However, even when using freedreno, non-free firmware is very likely still needed.

Generally speaking, Qualcomm devices have a lot of blobs and no modem isolation which is the reason why no Qualcomm-based device is yet supported by Replicant. See TargetsEvaluation for some analysis. We have not yet identified a Qualcomm-based device that would be a promising target for Replicant and where freedreno could be used on.

Vivante GCxxxx with Etnaviv

Not used by any supported device.

- Project page: https://github.com/etnaviv
- Source code: https://gitlab.freedesktop.org/mesa/mesa/-/tree/master/src/gallium/drivers/etnaviv

Etnaviv is a driver for some Vivante GCxxxx GPU variants based on Mesa's Gallium3D. It would be interesting to investigate whether devices using such a GPUs are good Replicant targets. We also have check if non-free firmware need to be loaded to the GPU.

kmsro

kmsro (kernel mode-setting render-only) is a Mesa's component that glues display-only (kms) with render-only drivers. It allows PRIME buffer sharing between these two counterparts.
It can also allocate scanout buffers using the display-only driver, catering for cases when only the display driver can allocate the appropriate memory (e.g. contiguous memory). These buffers are then shared through PRIME and imported at the render-only driver (e.g. lima_bo.c).

Background

The graphics cards present on most desktop and laptop computers include both a computing unit (GPU) and display controller circuitry, i.e., the display is directly connected to the graphics card. However, on the SoCs embedded on most mobile devices, these two functionalities are split between two different hardware components, there is:

- a render-only GPU, with no connections for displays;
- a display controller, where the displays are attached, that has no computing capabilities except for hardware compositing (blending of images/planes).

These two components then share image buffers through the main memory (RAM).
On the Linux kernel DRM subsystem, these appear as independent devices, handled by different drivers.

Along the years, Mesa got support for several render-only GPUs, through Gallium drivers like Lima, Panfrost, Freedreno and Etnaviv. The GPUs handled by these drivers are available on many different SoCs, with varying display controllers: exynos, imx, armada, etc. The number of possible combinations, and thus code duplication, grew in Mesa. kmsro was then introduced as an abstraction layer to get rid of this code duplication.

Gralloc

In order to have a working software rendering on Replicant 10 we need a gralloc (graphics memory allocator) library that:

- Implements the Android Gralloc HAL API version 1.
- Is compatible with drm-hwcomposer (a libre implementation, based on Linux's DRM, of Android's Hardware Composer HAL).
- Is compatible with Mesa and particularly its llvmpipe software renderer and the Lima driver.
- (optional) Is compatible with SwiftShader.

There are 3 free-software grallocs available:

- drm_gralloc - Directly uses Linux DRM for buffer allocation. Built by the Android-x86 team and now getting phased out in favor of gbm_gralloc.
- gbm_gralloc - Uses Mesa's GBM (Generic Buffer Management) for buffer allocation through libgbm. GBM itself will then call DRM.
- minigbm - Uses its own embedded GBM implementation. Does not depend on Mesa. Built by Intel for their Android-IA project now dubbed Project Celadon. Also used by Google in ChromiumOS.

The table bellow summarizes the capabilities of these 3 gralloc implementations (sourced from the slides of Mauro's Rossi talk at XDC 2018):

project	API version	GEM/flink names	PRIME fd	binderization
drm_gralloc	0	Yes	incomplete	No
gbm_gralloc	0	N/A	Yes	Yes
minigbm	0, 1	N/A	Yes	Yes

Key:

- **project**: the implementation name.
- **API version**: lists the supported versions of Android's gralloc HAL.
- **GEM/flink names**: whether there is support for GEM (Graphics Execution Manager) object sharing through GEM names. GEM names are unique 32 bit integers, created via the flink operation, that point to a unique GEM object inside a DRM device. GEM names are mostly deprecated due to security reasons.
- **PRIME fd**: whether there is support for buffer sharing between different DRM drivers through PRIME, the successor of GEM names. PRIME allows the conversion of local GEM handles to DMA-BUF file descriptors and vice-versa, via the DMA Buffer Sharing API.
- **binderization**: whether there is support for using a binder, a structure that allows memory sharing between different processes on the same machine (for instance a Queue).

gbm_gralloc

gbm_gralloc is the current choice for Replicant 10 as it:

- interoperates nicely with both drm-hwcomposer and Mesa;
- is maintained upstream;
- is actively used by other projects such as Android-x86 and SPURV.

gbm_gralloc and kmsro

gbm_gralloc accepts a configuration property gralloc.gbm.device that should point to the DRI node to use for the memory operations. However gbm_gralloc may not use the actual node set on that property. gbm_gralloc uses Mesa's GBM, which in turn may use kmsro (when active). As explained above, kmsro has the ability to allocate memory buffers on both render nodes and display nodes, according the the intended buffer usage. As such, when kmsro is active, gbm_gralloc will just care about the driver name of the node set on gralloc.gbm.device, and will operate on either the driver's display (kms) node or it's render node.
For instance, on a device with the exynos DRM driver, which exposes both a kms node (e.g. /dev/dri/card0) and a render-only node (e.g. /dev/dri/renderD128), if we set gralloc.gbm.device=/dev/dri/renderD128, gbm_gralloc will actually use /dev/dri/card0 when scanout buffers are requested.

Composer

Non Android Hardware Composer HAL compatible

- libliftoff **- Eases the use of KMS planes.**

Wayland

Implementation	Architecture	Advantages	Disadvantages	Sustainability
SPURV	Android<->Hwcomposer <->Wayland		Probably too low in the stack, maybe should replace surfaceflinger ideally? => Do some benchmarks	